The Addition of Geolocation to Sensor Networks

Robert Bryce[1] and Gautam Srivastava[2]
[1]*Heartland Software, Ardmore, Canada*
[2]*Department of Mathematics and Computer Science, Brandon University, Brandon, Canada*

Keywords: MQTT, IoT, Networks, Protocols, Geolocation, Broker.

Abstract: Sensor networks are recently rapidly growing research area in wireless communications and distributed networks. A sensor network is a densely deployed wireless network of small, low cost sensors, which can be used in various applications like health, environmental monitoring, military, natural disaster relief, and finally gathering and sensing information in inhospitable locations to name a few. In this paper, we focus on one specific type of sensor network called MQTT, which stands for Message Queue Transport Telemetry. MQTT is an open source publisher/subscriber standard for M2M (Machine to Machine) communication. This makes it highly suitable for Internet of Things (IoT) messaging situations where power usage is at a premium or in mobile devices such as phones, embedded computers or microcontrollers. In its original state, MQTT is lacking the ability to broadcast geolocation as part of the protocol itself. In today's age of IoT however, it has become more pertinent to have geolocation as part of the protocol. In this paper, we add geolocation to the MQTT protocol and offer a revised version, which we call MQTT-G. We describe the protocol here and show where we were able to embed geolocation successfully.

1 INTRODUCTION

Today, due to the increased use of smartphones, push notification services are now commonly used (Thangavel et al., 2014a). Push notifications service keeps the device online for every certain communication cycle, and the server pushes the mess ages to each client when necessary. Compared to the polling method, push notification method was proved to be more efficient in battery and data consumption (Banks and Gupta, 2014). **MQTT** is a client-Server publish/subscribe messaging transport protocol, described in Figure 2. It is light weight, open, simple, and designed to be easy to implement by both publishers and subscribers alike. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (**M2M**) and Internet of Things (**IoT**) contexts where a small code footprint is required and/or network bandwidth is at a premium. As a well-known example, Facebook Messenger is based on MQTT (Lee et al., 2013). MQTT Protocol is suitable for implementing integrated Simple Notification Service (SNS) gateway servers which can merge different SNS protocols and OS into a unified single platform. Also, there is no restriction in messaging while using push notification services.

Some of the positive characteristics of MQTT are its light weight nature and binary footprint, which lead it to excel when transferring data over the wire. In comparison to well used protocols like Hypertext Transfer Protoccol (**HTTP**), it only has a minimal packet overhead. Another important aspect of MQTT is that it is extremely easy to implement on the client side. This fits perfectly for constrained devices with limited resources. Its ease of implementation was one of the goals that was met when MQTT was invented (Stanford-Clark and Hunkeler, 1999).

1.1 History

MQTT was invented by Andy Stanford-Clark (IBM) and Arlen Nipper (**Arcom**, now **Cirrus Link**) in 1999. Its initial use was to create a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over satellite connections (Stanford-Clark and Hunkeler, 1999). It was then updated to include Wireless Sensor Networks in 2008 (Hunkeler et al., 2008). In (Stanford-Clark and Hunkeler, 1999), the following goals were specified:

- Simple to implement
- Provide a Quality of Service Data Delivery
- Lightweight and Bandwidth Efficient

Bryce, R. and Srivastava, G.
The Addition of Geolocation to Sensor Networks.
DOI: 10.5220/0006921907620768
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 762-768
ISBN: 978-989-758-320-9

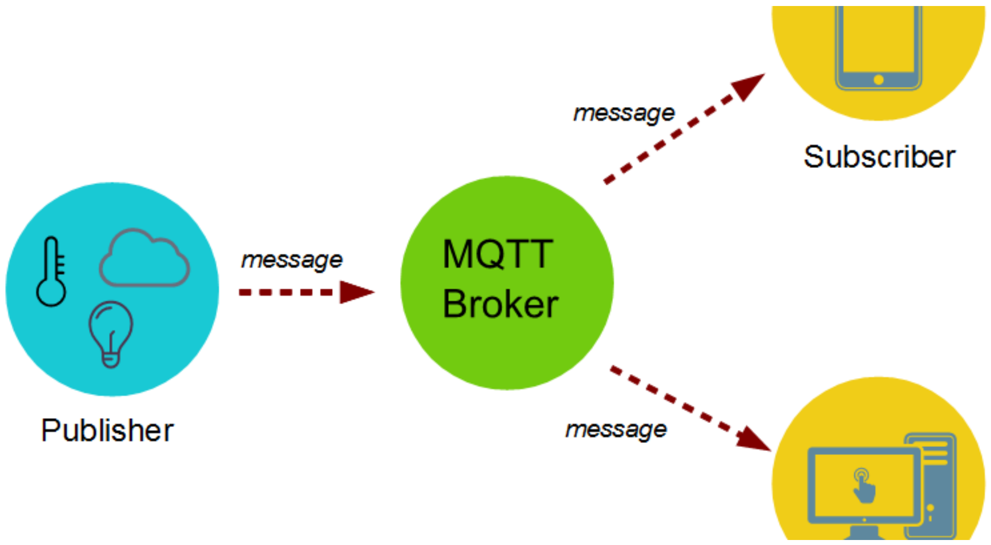

Figure 1: Publish and Subscribe Model of MQTT.

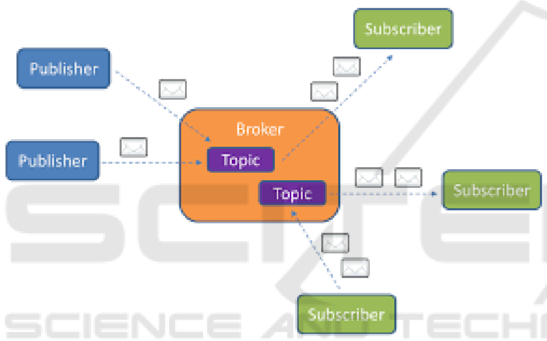

Figure 2: Publish and Subscribe Model of MQTT.

- Data Agnostic
- Continuous Session Awareness

We focus here on an open source implementation of **MQTT 3.1.1** called **Mosquitto**, prescribed recently in (Light, 2017). Mosquitto provides standard compliant server and client implementations of the MQTT messaging protocol.

To quote (Light, 2017),

> MQTT uses a publish/subscribe model, has low network overhead and can be implemented on low power devices such microcontrollers that might be used in remote Internet of Things sensors. As such, Mosquitto is intended for use in all situations where there is a need for lightweight messaging, particularly on constrained devices with limited resources.

In our current project, we focus on three parts, namely

- The main Mosquitto server
- The Mosquitto publish and subscribe client utilities
- An MQTT client library written in C/C++ wrapper

1.2 Current Uses of Mosquitto

We have seen some very interesting applications of MQTT recently using Mosquitto. First, (Thangavel et al., 2014b) compared the performance of MQTT and the Constrained Application Protocol (**CoAP**). CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks in IoT. The protocol is designed for M2M applications such as smart energy and building automation. In (Fremantle et al., 2014), the authors investigated the use of **OAuth** in MQTT. OAuth is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.

We have also seen Mosquitto used to evaluate MQTT for use in Smart City Services (Antonić et al., 2015). The authors compare MQTT and **CUPUS** in the context of smart city application scenarios, which is currently a hot topic with many large cities wanting to join the digital age and become Smart Cities. Furthermore, it has been used in the development of an environmental monitoring system (Bellavista et al., 2017). Mosquitto has also been used to support research less directly as part of a scheme for remote control of an experiment (Schulz et al., 2014).

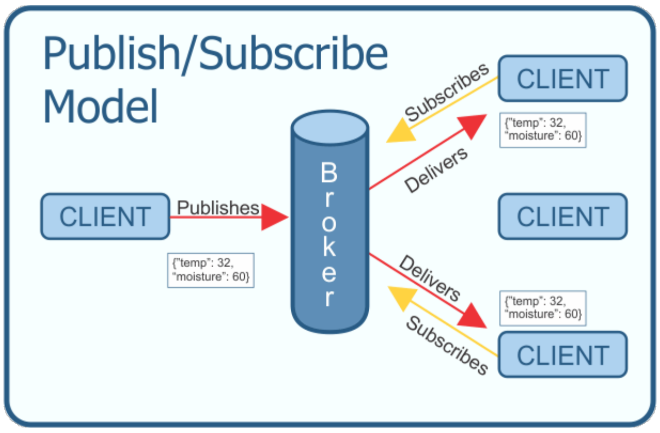

Figure 3: Publish and Subscribe Model of MQTT.

1.3 MQTT Publish/Subscribe Pattern

In a publish/subscribe pattern a client publishes information and other clients can subscribe to only the information they want. In many cases there is a broker between the clients who facilitates and/or filters the information. This allows for a loose coupling between entities.

The decoupling can occur in a few different ways: Space, Time, and Synchronization.

- **Space** - the subscriber does not need to know who the publisher is, for example by IP address, and vice-versa
- **Time** - the two clients do not have to be running at the same time
- **Synchronization** - Publishing and receiving does not halt operations

Through the filtering done by the broker not all subscribers have to get the same messages. The broker can filter on subject, content, or type of message. A client, therefore could subscribe to only messages about temperature data or only messages with content about centrifuge machines. They could only want to receive information about specific types of errors as well. In our version MQTT-G, we also add the ability for clients to receive messages based on some geolocation criteria.

1.4 Simple Example

Thinking about an IoT situation with, for example, a device with environmental sensors connected to it such as a temperature sensor and a moisture sensor, we could be publishing that data. Specific clients that are connected to our broker may be interested in that and others may not. They would subscribe to the information they want and the broker would provide the necessary information accordingly. The topic string is determinant regarding whether the data is forwarded to the subscriber or not. To date, we have seen lots of work on IoT with various uses and applications (Kopetz, 2011),(Weber and Weber, 2010), (Wortmann and Flüchter, 2015), and (Xia et al., 2012).

Once connected to the broker the publishing client simply sends its data to the broker. Once there, the broker relays the appropriate data onto the clients who have subscribed for that data. Again, those subscriptions can be filtered. The subscribing client then has the opportunity to use or discard the packet upon arrival. All of this data transferring is done in a light weight fashion designed for small, resource limited devices; network usage is efficient because logic on the broker's side provides the initial filtering.

The message packet shown in Figure 3 is just an example of what a message may look like. Along with the message, or payload, a real packet would include additional information such as a packet ID, topic name, and quality of service (QoS) level. Also included in the packet would be flags so the broker knows how long to retain the message and if the message is a duplicate.

This is just the tip of the iceberg for MQTT; there are several other features of interest as well. Features such as Retained Messages, Quality of Service, Last Will and Testament, Persistent Sessions, and SYS Topics to name a few. It should also come as no surprise given the importance of security in today's world, that MQTT has strong security features as well.

1.4.1 Our Contributions

We modify MQTT by adding geolocation information into specific MQTT packets such that, for example, client location could be tracked by the broker, and clients can subscribe based on geolocation. This

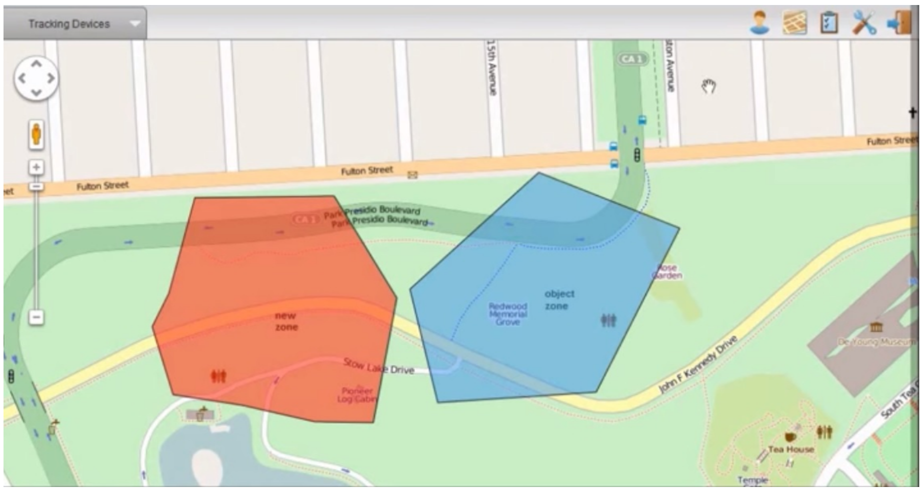

Figure 4: Polygon Geofence.

can lead to the clients last known location having a comparison to a **polygon geofence**. One of the important features of GPS Tracking software using GPS Tracking devices is geofencing and its ability to help keep track of assets. Geofencing allows users of a Global Positioning System (GPS) Tracking Solution to draw zones (GeoFence) around places of importance, customers sites and secure areas.

In MQTT-G, by adding geolocation, information reaching subscribers can be filtered out by the broker to only fall within the subscribers geofence. We can see an example of a geofence in Figure 4. Take for example a forest fire containment situation. By prescribing a geofence, specific subscribers can work to contain the fire knowing the polygon geofence in real time as given out by the messages from the brokers to clients and without receiving messages associated with a seperate geofence/forest fire. Furthermore, third-party additions to the containment of the fire could quickly be added to the subscriptions using the geofence. Other subscribers (managers) may choose geofence (being a political boundary) to monitor all messages associated with all geofences/forest fires in a given area. The applications for this are plenty and include:

- Field team coordination
- Search and rescue improvements
- Advertising notifications to customers within specific ranges
- Emergency notifications, such as inclement weather or road closures.

2 RESULTS

The basis of adding geolocation to MQTT is to leverage unused binary bin data within the protocol definition itself and optionally embedding geolocation data between the header and payload. The major change to the packets was the inclusion of the **Geolocation Flag**. The flag is sent in packets between the client to broker to notify the broker that a client is sending geolocation data in the packet. The packets that are used to send geolocation information are given in Table 1 derived from the original protocol implementation.

Geolocation is not sent for **CONNACK, SUBACK, UNSUBACK, PINGRESP** packets as they are only for information passed from broker to client, and thereby deemed unnecesary to contain geolocation information. For all packets mentioned in Table 1, with the exception of **PUBLISH**, the 3rd bit of the fixed header is unused (reserved) in the original implementation in (Stanford-Clark and Hunkeler, 1999), so we can easily use it to indicate the presence of geolocation information.

Figures 5 and 6 explain where the location data is on the packet. We also give the structure of the code shown by the struct **mosquitto_location**.

```
struct mosquitto_location {
uint8_t version;
double lat, lon;
float elev;
};
```

The PUBLISH control packet needs a different implementation. Because the 3rd bit is already allocated for Quality of Service (**QOS**), and all other pac-

Table 1: Packets Used For Geolocation.

Packet	Details
CONNECT	client request to connect to Server
PUBLISH	Publish message
PUBACK	Publish acknowledgment
PUBREC	Publish received (assured delivery part 1)
PUBREL	Publish received (assured delivery part 2)
PUBCOMP	Publish received (assured delivery part 3)
SUBSCRIBE	client subscribe request
UNSUBSCRIBE	Unsubscribe request
PINGREQ	PING request
DISCONNECT	client is disconnecting

kets are also reserved for an existing use, we chose to implement a new control packet type. **PUBLISHg** (=0xF0) is used as the flag type for geolocation data when it is to be sent. There are 16 available command packet types within the MQTT standard and 0 through 14 are used.

We deem geolocation data as an optional attribute, as not all clients may wish to publish any geolocation data. The geolocation of manager of the forest fire crews in the aforementioned example does not need to be shared with the crews, it is irrelevant. In our approach, geolocation data is not included in the packet payload, since not all packet types support a payload, thus rendering payloads not a viable option. Furthermore, we did not wish to require the broker to examine the payload of any packet, thus keeping our processing footprint low.

2.1 Justification of Overhead

There are many transport protocols that MQTT packets may go over in size, where 21 bytes may be a concern. However, if you are using TCP/IP as we discuss here, 21 bytes is negligible when considering the bytes required just for the overhead of TCP/IP. Finally, there is no need for a client to only submit MQTT-G (geolocation) packets; there is nothing stopping a client to submit MQTT-G packets when it is time to update location, then just use unmodified MQTT packets otherwise. The logic is based on a single bit, and we have not introduced any requirement that all packets must be MQTT-G either from a specific client or from multiple clients.

2.2 Handling of Packets

Packets that are received without geolocation are handled via the original Mosquitto functions, and as such can be left unmodified. Packets that are received with geolocation are handled similarly but with a call to a **last known location** updating method, which stores the clients unique ID and the location data into a **Hashtable** object designed to be compared against the geofence if and only if they are a subscriber to be sent a PUBLISH. We have elected to attach geolocation data from all packet types originating from the client to eliminate the need for specific packets carrying only geolocation data, and thus reducing overall network traffic as well.

2.3 Quality of Service

To ensure the reliability of messaging, MQTT supports 3 levels of Quality of Services(QoS) (Behnel et al., 2006). Figure 7 shows packet exchange measure according to 3 different QoS levels. QoS Level 0 sends message only once following the message distribution flow, and does not check whether the message arrived to its destination. Therefore, in case of sizable messages, it is possible that the message will be lost when any kind of loss comes in the way. QoS Level 1 sends the message at least once, and checks the delivery status of the message by using the status check message, PUBACK. However, when PUBACK is lost, it is possible that the server will send the same message twice, since it has no confirmation of the message being delivered. QoS Level 2 passes the message through exactly once utilizing the 4-way handshake. It is not possible to have a message loss in this level, but due to the complicated process of 4-way handshake, it is possible to have relatively longer end-to-end delays. The higher QoS level is the more packets will be exchanged.

It is true that higher-level QoS is more effective if you do not want any message loss, but such complicated processes will increase the end-to-end delay. If we can deduct appropriate QoS level utilizing correlation analysis of the relationship between the message loss due to the size of payload and the end-to-end delay, it will be possible to build a optimal service network for push notification services.

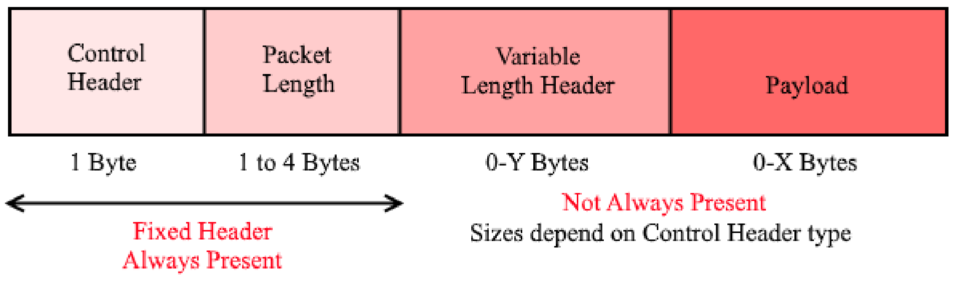

Figure 5: Original MQTT Packet.

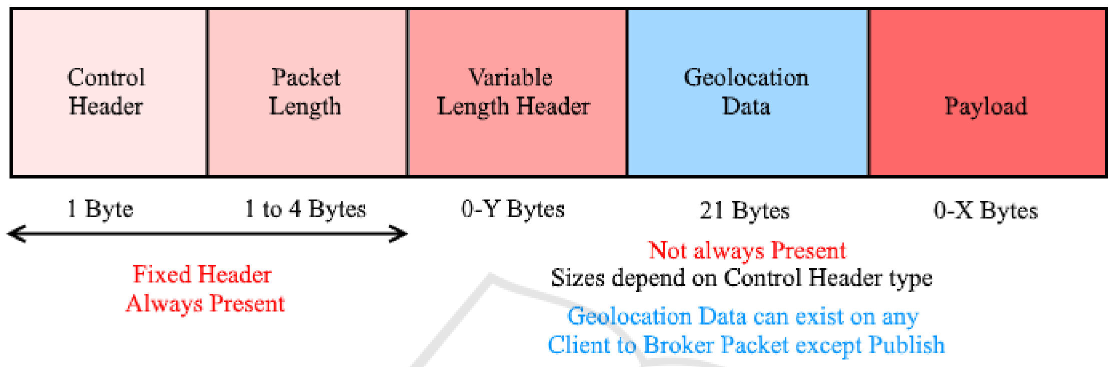

Figure 6: MQTT Geolocation Packet.

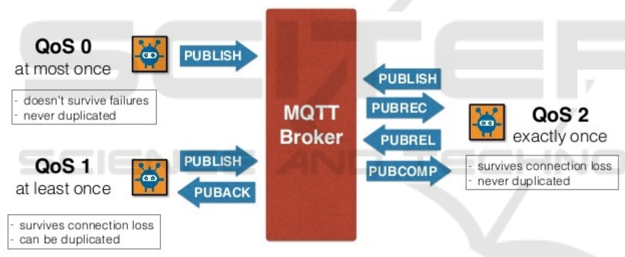

Figure 7: Quality of Service.

2.4 Geofencing

Creating the geofence code was a major part of the addition of geolocation to MQTT. The geofence filtering is only called when a PUBLISH reaches the broker, as these packets are forwarded to subscribing clients.

The **mosquitto_check_polygon** returns a boolean value indicating whether the clients last known location is within the polygon. If the point is outside the polygon, it simply aborts forwarding the PUBLISH as the client has indicated it is not interested in the message. This condition is tested for each client so that other subscribers may receive packets of interest. Thus, we have used our own custom geometry library originally implemented in (Tymstra et al., 2010) with features first discussed in (Bose et al., 2009). The library is unmodified for the broker implementation, but it is reconfigured for mobile clients.

Geofence data is presently submitted and cleared by a client to the broker using the **$SYS** MQTT topic convention so that clients may individually submit geofences of interest. The broker maintains polygon data for each subscribing client. Polygons may be *static* in shape and location or *dynamic* and move with the last known location of the client.

3 FUTURE WORK

We are still finishing the final testing of the implementation of MQTT-G for the Android OS. Once finished, the implementation pull geolocation directly from the Android OS and allow clients to quickly and easily create and destroy polygon geofences on the go. Applications of the Android client are plentiful but have some key uses in natural disaster containment and safety in this age of mobile devices and Internet of Things.

One alternative approach that we have not explored is introducing geolocation data to the MQTT payload. However, in this instance, the packet would need to be forwarded to a specific client understanding the format of the payload, then sending the PUBLISH data on another topic for interested clients to subscribe to. This division of logic between a separate client and the broker potentially doubles the load on the broker and is thus not considered. However, geofencing remains an expensive operation in comparison to the rest of the broker logic, and future versions of the broker will be multithreaded to address performance concerns.

4 CONCLUSION

MQTT is an open source standard for M2M communication. Originally designed by IBM, the main use for MQTT is as a publisher/subscriber protocol. In this paper, we have introduced a new version, called MQTT-G, that adds geolocation information to the protocol and offers a revised implementation, that can help aid in the breadth of uses for MQTT. We also modernize the protocol to include a somewhat standard feature of most protocols in today's IoT age. The advanced protocol we implement can be used to offer geolocation as part of the publish/subscribe infrastructure, thus aiding in the real time applications that it can be used for. Our implementations offer versions for both its native C/C++ environment and as a mobile Android client as well.

REFERENCES

Antonić, A., Marjanović, M., Skočir, P., and Žarko, I. P. (2015). Comparison of the cupus middleware and mqtt protocol for smart city services. In *Telecommunications (ConTEL), 2015 13th international conference on*, pages 1–8. IEEE.

Banks, A. and Gupta, R. (2014). Mqtt version 3.1. 1. *OASIS standard*, 29.

Behnel, S., Fiege, L., and Muhl, G. (2006). On quality-of-service and publish-subscribe. In *Distributed Computing Systems Workshops, 2006. ICDCS Workshops 2006. 26th IEEE International Conference on*, pages 20–20. IEEE.

Bellavista, P., Giannelli, C., and Zamagna, R. (2017). The pervasive environment sensing and sharing solution. *Sustainability*, 9(4):585.

Bose, C., Bryce, R., and Dueck, G. (2009). Untangling the prometheus nightmare. In *Proc. 18th IMACS World Congress MODSIM09 and International Congress on Modelling and Simulation, Cairns, Australia*, pages 13–17.

Fremantle, P., Aziz, B., Kopeckỳ, J., and Scott, P. (2014). Federated identity and access management for the internet of things. In *Secure Internet of Things (SIoT), 2014 International Workshop on*, pages 10–17. IEEE.

Hunkeler, U., Truong, H. L., and Stanford-Clark, A. (2008). Mqtt-sa publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE.

Kopetz, H. (2011). Internet of things. In *Real-time systems*, pages 307–323. Springer.

Lee, S., Kim, H., Hong, D.-k., and Ju, H. (2013). Correlation analysis of mqtt loss and delay according to qos level. In *Information Networking (ICOIN), 2013 International Conference on*, pages 714–717. IEEE.

Light, R. A. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13).

Schulz, M., Chen, F., and Payne, L. (2014). Real-time animation of equipment in a remote laboratory. In *Remote Engineering and Virtual Instrumentation (REV), 2014 11th International Conference on*, pages 172–176. IEEE.

Stanford-Clark, A. and Hunkeler, U. (1999). Mq telemetry transport (mqtt). *Online]. http://mqtt. org. Accessed September*, 22:2013.

Thangavel, D., Ma, X., Valera, A., Tan, H.-X., and Tan, C. K.-Y. (2014a). Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE.

Thangavel, D., Ma, X., Valera, A., Tan, H.-X., and Tan, C. K.-Y. (2014b). Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE.

Tymstra, C., Bryce, R., Wotton, B., Taylor, S., Armitage, O., et al. (2010). Development and structure of prometheus: the canadian wildland fire growth simulation model. *Natural Resources Canada, Canadian Forest Service, Northern Forestry Centre, Information Report NOR-X-417.(Edmonton, AB)*.

Weber, R. H. and Weber, R. (2010). *Internet of things*, volume 12. Springer.

Wortmann, F. and Flüchter, K. (2015). Internet of things. *Business & Information Systems Engineering*, 57(3):221–224.

Xia, F., Yang, L. T., Wang, L., and Vinel, A. (2012). Internet of things. *International Journal of Communication Systems*, 25(9):1101.